Asynchronous Invocation of Adaptations in Electronic Structure Calculations

Sai Kiran Talamudupula, Masha Sosonkina
Ames Laboratory, USDOE, Iowa State University
{kiran123, masha}@scl.ameslab.gov

Michael W. Schmidt
Department of Chemistry, Iowa State University
mike@si.msg.chem.iastate.edu

Abstract

Modern quantum chemistry deals with electronic structure calculations of unprecedented complexity and accuracy. They demand full power of high-performance computing and must be in tune with the given architecture for superior efficiency. To make such applications resource-aware, it is desirable to enable their static and dynamic adaptations using some external software (middleware), which may monitor both system availability and application needs, rather than mix science with system-related calls inside the application.

This paper investigates models of application interlinking with middleware based on the example of the computational chemistry package GAMESS and middleware NICAN. The existing synchronous model is limited by the possible delays due to the middleware processing time under the sustainable runtime system conditions. Proposed asynchronous and hybrid models aim at overcoming this limitation. The experiments contribute to gaining an insight into a choice of the model.

Keywords: quantum chemistry, GAMESS, application adaptations, middleware, system resource congestion

1. INTRODUCTION

Electronic structure calculations provide a representative example of High-Performance Computing (HPC) applications. They require a large amount of disk space—typically, several gigabytes—to store the integrals and a large I/O bandwidth to use them during the iterative solution process. The need for the CPU and memory resources scales up with the number of atoms as a power of 4–7 and 2–4, respectively, depending on the type of calculation executed. Thus, pooling all the memory and disk resources available in distributed environments emerges as the solution to such an explosion in computational needs and, at the same time, challenges to obtain an optimal parallel performance. The latter is nearly infeasible without an application awareness of parallel architecture, that is, without being tuned to the architecture at hand.

Dynamic adaptations respond in a timely manner to runtime changes in the environments, as described, e.g., in [7]. To implement such adaptations, an application may be supplied with sophisticated system monitoring strategies and/or with the analysis of its own (historical) performance. The most non-intrusive and portable way to accomplish this is to empower an application with a middleware that will provide a liaison between the application and system by monitoring the system resources dynamically, making adaptation decisions based on the application performance, and invoking application adaptations if needed.

For the computational chemistry, adaptive software architecture is an emerging area of research. Existing architectures vary from compiler- and model-based engineering approaches to adaptive algorithms. In [3], a component-based approach is described. It interfaces many independently developed numerical adaptation algorithms and implementations. A recent development has been the inclusion of support for CQoS, computational quality of service, which has a measurement, analysis, and control infrastructure for dynamic domain-specific decision making. An integration of HPC applications with middleware tools may empower the applications with such advanced features as parallel and distributed programming models, system resource management, remote application monitoring, and user-interactive material. Middlewares are more effective if they provide these attractive features dynamically.

Active Harmony [11] is an infrastructure in which the application acts as a client when sending information to the server, which, in turn, makes application tuning and adaptation decisions based on this information from the client. The middleware system dQUOB [5] provides continuous evaluation of queries over time sequenced data. The queries are inserted into the data streams at run time and managed remotely during the execution. The goal of IANOS [13] is to provide a general middleware infrastructure that allows optimal positioning and scheduling of HPC Grid applications. The NICAN middleware, proposed in [9], has been extensively used as adaptive mechanism invocation tool in quantum chemistry applications, such as GAMESS [1]. The interaction between NICAN and GAMESS has not been analyzed yet, however. Modeling GAMESS execution with and without NICAN middleware prompted to consider an asynchronous mode of their interaction that appears beneficial for lightly oscillating system conditions when only few, if any, application adaptations are required.

This paper is organized as follows. Section 2. gives an overview of the computational chemistry package GAMESS and discusses its combination with a middleware tool NICAN. Section 3. extends the integration to the asynchronous model, followed by its comparison with the synchronous one. Note that, in the rest of the paper, the term "integration" refers to a combination of the application and middleware rather than to numerical integration, which is a stage in quantum chemistry calculations. Performance results and relevant discussions are provided in Section 4.. Section 5. concludes and notes on the future work.

2. GAMESS-NICAN INTEGRATION

GAMESS is a computational chemistry application which is widely used to perform *ab-initio* molecular quantum chemistry calculations. A wide range of quantum chemistry computations are possible using GAMESS such as calculating Restricted Hartree-Fock (RHF) Self-Consistent Field (SCF) molecular wavefunctions. Using the SCF method, GAMESS iteratively approximates solution to the Schrödinger equation that describes the basic structure of atoms and molecules. SCF is one of the many computationally intensive parts of GAMESS and has two different implementations (i.e., execution modes), *direct* and *conventional*, which differ in the way two-electron (2-e) integrals are computed. In the direct mode, 2-e integrals are recomputed "on-the-fly" for each iteration consuming mainly physical memory and CPU resources. In the conventional, these integrals are calculated before the first SCF iteration and stored in a file on disk. The subsequent iterations fetch the 2-e integrals from disk, consuming mainly I/O bandwidth. So, there are instances when the conventional SCF implementation proves to be better compared to the direct one and vice-verse. It is feasible to alternate between two available implementations at run-time. Therefore, by integrating the SCF process with a middleware that interacts with the system, dynamic adaptations of SCF may be enabled. There are other (higher level of theory) calculations available in GAMESS, such as the second order Möller-Plesset correction (MP2), that are used to get electron correlations and involve post SCF calculations to improve the accuracy of solution. For more details on GAMESS see [1].

2.1. Using NICAN

In [8, 6, 12], an integration of GAMESS with NICAN has been discussed for the purpose of GAMESS performance tuning and prediction. One way to use NICAN with GAMESS is to enable GAMESS adaptations in the computationally intensive SCF method. Specifically, NICAN prompts GAMESS to switch from one SCF implementation to another at the iteration following the detection of the decrease in the GAMESS performance as measured by its execution time. An increase in the iteration time typically signifies a change in system conditions. The NICAN architecture, containing a manager dedicated to the application along with the various types of modules to collect the system information and to encapsulate adaptation control, makes possible this switch. The timely nature of adaptations is assured by GAMESS waiting after each iteration for the decision from NICAN. Fig. 1 depicts this GAMESS-NICAN integration featuring a separate `GMS_NCN` module that implements the adaptation control mechanism and the GAMESS-NICAN synchronization (depicted as lines with double arrows) to communicate the adaptation decision from NICAN to GAMESS. In general, NICAN consists of dynamically loadable modules making it versatile and providing a wide variety of interactions with system or application [4, 10].

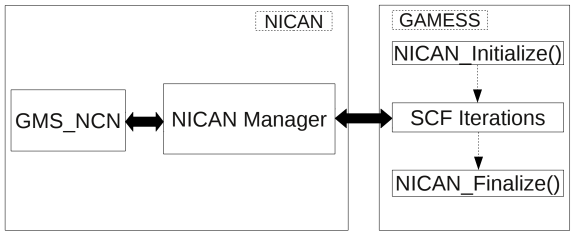

Figure 1. GAMESS-NICAN integration in the synchronous model.

3. ASYNCHRONOUS INTEGRATION

Timely adaptations are an important advantage of the GAMESS-NICAN integration with synchronization, but they come at a price: When no or few adaptations are required the GAMESS-NICAN execution time may suffer. Since a certain system condition typically persists in supercomputers executing HPC applications, it is worth considering an *asynchronous* integration model. This model differs from its synchronous counterpart in that an application proceeds to the next iteration without waiting for the decision from NICAN and adapts, if necessary, later. Certainly, delaying adaptations may be detrimental to the overall performance. Using GAMESS as an example, this section investigates when the delays may be tolerated.

Assume that the execution time of a single iteration, in either direct or conventional mode, does not deviate much from an average iteration time for that mode in a certain system state. Consider two distinct system states, Congested C and Free F, and two SCF execution modes conventional c and direct d. Then, denote a single iteration time as t_m^S, where $m \in \{d,c\}$ and $S \in \{C,F\}$. The execution times T^S and $\overline{T}^S$ for the total N iterations when integrated with NICAN synchronously and asynchronously, respectively, may be expressed as

$$T^C = \sum_{j=1}^{K+1} \delta_j t^C_{m_j} + N\tau\,, \quad (1)$$

$$T^F = N t^F_m + N\tau\,, \quad (2)$$

$$\overline{T}^C = \sum_{j=1}^{\overline{K}+1} \overline{\delta}_j t^C_{m_j}\,, \quad (3)$$

$$\overline{T}^F = N t^F_m\,, \quad (4)$$

where τ is the time required to communicate with NICAN including the time to decide on an adaptation. K and $\overline{K}$ are the number of adaptations for synchronous and asynchronous models, respectively, such that $K \geq \overline{K}$ and $j = 0$ at the first SCF iteration while $j = K + 1$ happens at the last (Nth) iteration. Then, δ_j and $\overline{\delta}_j$ are the numbers of iterations between adaptation j and $(j-1)$ for the synchronous and asynchronous models, respectively.

The expression (4) of $\overline{T}^F$ contains no term related to NICAN. Thus, this total time is the same as when GAMESS is executed without NICAN in the non-congested environment, and the integration with the NICAN tool becomes less intrusive.

To analyze the effect of delayed adaptations, consider that δ_1 is always less than $\overline{\delta}_1$. However, for HPC applications, such as GAMESS, it may be assumed that the adaptation decision arrives when only one more GAMESS iteration elapses, i.e., an adaptation is delayed by only one iteration. Hence, $\overline{\delta}_1 - \delta_1 = 1$. An assumption of *persistent congestion* also follows from the fact that an HPC application may heavily consume supercomputer resources, so they are not acquired or released momentarily. Here, persistent congestion is defined as a system resource congestion the detection of which causes the given application to adapt *only once* since the congestion continues to manifest itself until the application termination. This assumption may be reasonable in the adaptive application parts that are short relative to the entire application execution time. For example, SCF iterations are typically much faster than the higher-level calculations, such as MP2. Under these two additional assumptions, the equations (1) and (3) may be rewritten as

$$T^C = \delta_1 t^C_{m_1} + (N - \delta_1) t^C_{m_2} + N\tau\,, \quad (5)$$

$$\overline{T}^C = (\delta_1 + 1) t^C_{m_1} + (N - \delta_1 - 1) t^C_{m_2}\,. \quad (6)$$

The asynchronous model becomes beneficial when the total time $\overline{T}^C$ is smaller than T^C, i.e., when

$$t^C_{m_1} - t^C_{m_2} < N\tau\,. \quad (7)$$

In other words, the time difference between iterations in different SCF execution modes under the persistent congestion should be smaller than the NICAN execution time for the total N iterations of SCF. This may happen when either N or τ are very large since there is not much sense to adapt if the difference $t^C_{m_1} - t^C_{m_2}$ is close to zero.

To increase the number of cases when the asynchronous integration may be applied, a hybrid model is developed under the same assumptions. It behaves as the synchronous one until the adaptation is invoked and as asynchronous thereafter. For the hybrid model, the total SCF iteration time $\widetilde{T}^S$ is given as

$$\widetilde{T}^C = \delta_1 t^C_{m_1} + (N - \delta_1) t^C_{m_2} + \delta_1 \tau\,, \quad (8)$$

$$\widetilde{T}^F = N t^F_m + N\tau\,. \quad (9)$$

Note that the first two terms of equation (8) come from (6) for the synchronous model while the third term in (8) is that of (6) being reduced by $(N - \delta_1)\tau$. Thus, the difference between $\widetilde{T}^C$ and T^C is always negative, which implies that the hybrid integration model performs better than the synchronous. On the other hand, in the no congestion system state F, both models incur the same overhead.

3.1. Implementation details

The data regarding the iteration time is collected and used by NICAN to make adaptation decisions. NICAN checks if the iteration time is above a certain limit and decides on the execution mode switch. Following the presentation in [6], the pseudo-code for the jth adaptation decision may be written as:

```
$t_i$ = Actual time taken for iteration $i$
$t^u$ = Upper bound for the time per iteration
$m$ = Average iteration time between two adaptations
$t^e$ = Estimated run-time for a single iteration (obtained
by NICAN after running a GAMESS "check" run first.)
$\Delta_i = |t^e - t_i|$
if $t_i > t^u$ OR $t_i > m + \Delta_i$ then
  if (SCF is conventional) then
    switch to direct
  else if ((no peer conventional jobs) then
        switch to conventional
```

The asynchronous model requires an instant return of control to GAMESS after communicating with the middleware. The synchronous model is implemented via Sockets API for the underlying (blocking) communication and can be modified into the asynchronous one by changing the communication to non-blocking. The NICAN Manager-Module interaction is the base for these changes and uses multi-threading. The Manager invokes the module thread and passes to it the requests from GAMESS. Due to the asynchronous behavior of thread scheduling in NICAN and of integration with GAMESS, race conditions are possible and NICAN "stale"

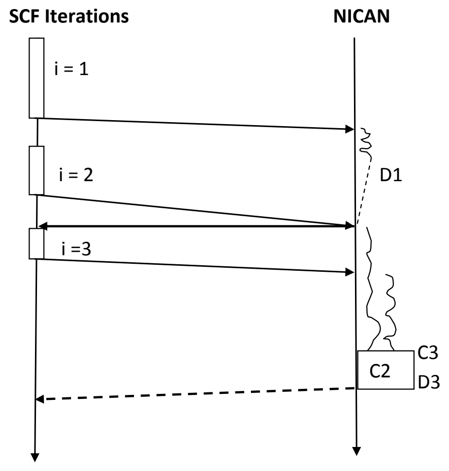

Figure 2. Timeline diagram indicating the temporal dependence between SCF iterations and NICAN.

decisions may arise. The race conditions are taken care of by encapsulating the NICAN control algorithm into a critical section executed sequentially by the module threads. To eliminate the production of stale decisions, a global data structure, represented by an array of the size equal to the number of iterations *N*, has been implemented. Since a stale decision should not be sent to GAMESS, before completing its work a given thread checks whether there is no other thread with a larger number waiting to enter the critical section.

Fig. 2 shows a timeline diagram for the GAMESS-NICAN interaction while depicting the handling of possible stale results of the decision-making process in the module. `D1` is the decision made by the fist thread causing adaptation in the third SCF iteration. A rectangular box next to the timeline of NICAN module represents the critical section encapsulating the control algorithm. The box interior (`C2`) shows that the second thread has entered the critical section and is ready to make a decision while the third thread (`C3` outside the box) is waiting for a lock. To avoid stale results, the second thread is preempted from the critical section execution and has to release the lock for the thread. Thus, the decision `D3` will be made by the third thread and passed to GAMESS (dashed arrow) in this scenario.

The hybrid model of execution starts as the synchronous model and shifts to the asynchronous one when the adaptation occurs once, which is marked by the global variable maintaining the SCF execution mode.

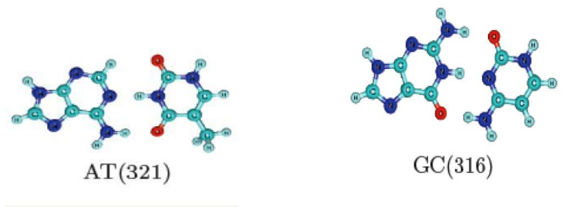

Figure 3. Molecule structure: AT (left) and GC (right)

4. EXPERIMENTAL RESULTS

To obtain performance results, two molecules Adenine-Thymine DNA base pair (AT) and Guanine-Cytosine DNA base pair (GC) are chosen (Fig. 3). AT and GC are represented using 321 and 316 basis functions, respectively. The tests were conducted for two different input combinations: AT is treated by SCF only while GC continues to the second order Möller-Plesset (MP2) after performing the SCF iterations, thus giving a more accurate energy value. The SCF RHF iterations were performed for each combination. Both molecules start in the conventional mode, i.e., store the 2-e integrals gin a file on disk and fetch the integrals from the file when required.

4.1. Architecture and software used

The Dynamo cluster, located in the Scalable Computing laboratory, is dedicated to GAMESS and other HPC application research. There are 35 Intel Xeon E5420 nodes with two quad-core processors running at 3 GHz. Each node has a RAM of 16 GB, a 1.5 TB scratch space and runs a 64-bit Redhat Linux. For the experiments presented here, the GAMESS jobs integrated with NICAN are run on up to five entire compute nodes of the cluster, i.e., on 8, 16, 24, 32, 40 cores.

All the three models, synchronous, asynchronous, and hybrid, are considered for the integration of parallel GAMESS calculations and the middleware tool NICAN. Resource congestion is introduced in all the nodes using a file-system benchmarking utility IOzone [2] during the GAMESS run, such that the congestion persists until the GAMESS completion. NICAN is assumed to take 10 seconds to decide on one adaptation. For the experiments reported here, the smallest difference between executing any two SCF iterations in two different modes was observed to be 160 seconds. This value is greater than the time $N\tau$ consumed by NICAN for $N = 15$ iterations typically allocated to SCF and $\tau = 10$ seconds. Furthermore, the largest difference between any two adjacent SCF iterations was observed to be about 199 seconds. So, the NICAN execution time was deliberately chosen to demonstrate how the asynchronous model behaves under the unfavorable conditions and why the hybrid model may be

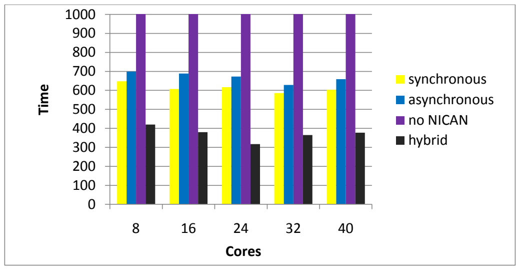

Figure 4. AT molecule input with synchronous, asynchronous, hybrid integration, and without NICAN in the presence of persistent congestion.

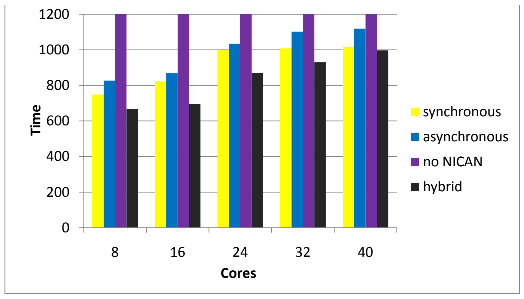

Figure 5. GC molecule input with synchronous, asynchronous, hybrid integration, and without NICAN in the presence of persistent congestion.

needed.

4.2. Discussion of results

For the AT and GC molecules in the presence of congestion, the GAMESS performance is shown in Fig. 4 and Fig. 5, respectively, with the synchronous, asynchronous, and hybrid NICAN integration models, as well as when no middleware is used. The number of cores is represented on the X-axis whereas the wall-clock time in seconds is shown on the Y-axis. The congestion forces an adaptation as decided by the control algorithm (Section 3.1.).

SCF iterations, starting in the conventional mode and running in the presence of congestion (disk I/O), benefit when they use synchronous model rather than asynchronous model of integration. This gain can be attributed to the timely avail-

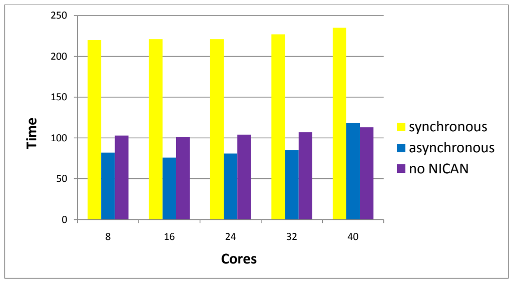

Figure 6. AT molecule input for synchronous, asynchronous, no-middleware integration in absence of congestion.

ability of the adaptation decision given by NICAN and the adaptation happening promptly in the iteration following the congestion discovery. The asynchronous model of integration is outperformed slightly by the synchronous one due to the additional iteration executing in the conventional mode. For both molecules (Fig. 4 and Fig. 5), the number of adaptations is one, i.e., $K = \overline{K} = 1$. The numbers of iterations δ_1 and $\overline{\delta}_1$ before the adaptation equal to one and two for the synchronous and asynchronous models, respectively, while the iteration numbers equal to 14 and 13 after the adaptation.

Even with the adaptation delayed, the asynchronous GAMESS-NICAN integration outperforms GAMESS running without adaptations as depicted by the tallest bars in Fig. 4 and Fig. 5. The "no-NICAN" case runs much slower than 1,000 and 1,200 seconds serving as a cut-off in Fig. 4 and Fig. 5, respectively, to preserve a good exposition scale. A key observation is that, the workloads as well as dynamic resource conditions contribute to adaptations at the run-time, and thus, they both account for the number and frequency of execution mode shifts. As mentioned earlier, the persistent congestion limits the number of adaptations to one.

Consider the hybrid model of integration, which is a mixture of synchronous and asynchronous ones. This model eliminates adaptation delays for the SCF iterations—thus, alleviating the drawbacks of the asynchronous model—by remaining in the synchronous state while expecting an adaptation and then switching to the asynchronous one with an assumption that the triggering event (i.e., congestion) will persist. This switch improves the GAMES-NICAN performance by eliminating subsequent synchronizations. As predicted theoretically in Section 3., the hybrid model performs better than the synchronous one under this assumption.

The wall clock times of the GAMESS runs without disk I/O congestion for synchronous and asynchronous models and

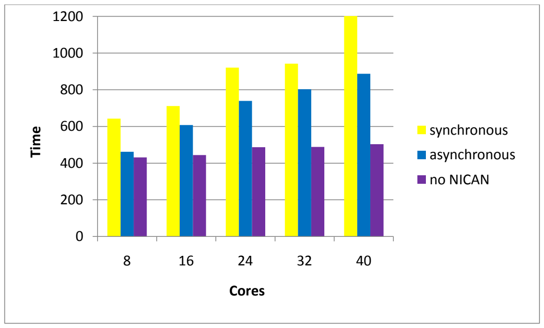

Figure 7. GC molecule input for synchronous, asynchronous, no-middleware integration in absence of congestion.

Table 1. MP2 electron correlation and SCF as percentages of the total execution time for GC in the absence of congestion and no NICAN.

Cores	8	16	24	32	40
MP2, %	74.4	80.4	86.4	86.6	85.8
SCF, %	15.8	17	12.5	12.3	13

without middleware integration for the AT and GC molecules are shown in Fig. 6 and Fig. 7, respectively. When there is only a single conventional job per node, i.e., there is no heavy disk I/O congestion, the synchronous model of integration makes computationally intensive SCF calculations wait for the return of NICAN. In the absence of congestion, the hybrid model reduces to the synchronous model, since there is no call for the adaptation. This may be viewed as the drawback of the hybrid model. The asynchronous model of integration alleviates this situation since it returns immediately to continue with the subsequent SCF iterations. Thus, for the AT and GC (Fig. 6 and Fig. 7), the asynchronous model delivers almost the same performance as GAMESS without NICAN in the environment requiring no adaptations.

In the case of GC, an MP2 calculation is used after the SCF iterations complete and takes a significant percent of the execution time as shown in Table 1. Therefore, the gains over the synchronous execution are less pronounced. Also, note that, for the GC calculation (Fig. 7), the wall clock time does not decrease with the increase in the core numbers, which may be attributed to the problem size not being sufficiently large to outweigh the communication and other parallel overhead.

5. CONCLUSIONS

This work investigates three integration models—synchronous, asynchronous, and hybrid—of the NICAN middleware with computationally-intensive applications, such as GAMESS. The total execution times have been compared among these models when GAMESS algorithm (SCF) adaptations are invoked and when no adaptation is needed. It has been demonstrated theoretically and supported by the experiments that, under the assumption of persistent congestion, the hybrid model outperforms the synchronous one, which, in turn, performs better than asynchronous. However, it may be detrimental to decide on the type of integration statically, before the actual execution, since the the presence or absence of congestion is typically a runtime system condition. Therefore, a hybrid model of integration is a better choice to start with when the system conditions may not be ideal and a congestion is probable.

As a future work, the hybrid model is to be extended for the multiple congestion phases and applied in other computationally-intensive parts of electronic structure calculations as well as in other applications that have been integrated with NICAN.

ACKNOWLEDGMENTS

This work was supported in part by Iowa State University under the contract DE-AC02-07CH11358 with the U.S. Department of Energy, by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC02-05CH11231, and by the National Science Foundation grants NSF/OCI – 0749156, 0941434, 001047772.

REFERENCES

[1] Baldridge, K. K., Boatz, J. A., Elbert, S. T., Gordon, M. S., Jensen, J. H., Koseki, S., Matsunaga, N., Nguyen, K. A., Su, S., Windus, T. L., Dupuis, M., Jr, J. A. M., and Schmidt, M. W. General atomic and molecular electronic structure system. *Journal of Computational Chemistry, November 1993* (1993), 1347–1363.

[2] IOzone filesystem benchmark. http://www.iozone.org/.

[3] Kenny, J. P., Wu, M., Huck, K., Gaenko, A., Gordon, M. S., Janssen, C. L., McInnes, L. C., Mori, H., Netzloff, H. M., Norris, B., and Windus, T. L. Adaptive application composition in quantum chemistry. In *The 5th International Conference on the Quality of Software Architectures (QoSA 2009), East Stroudsburg University, Pennsylvania, USA, June 22-26, 2009* (2009).

[4] Kulkarni, D., and Sosonkina, M. A framework for integrating network information into distributed iterative solution of sparse linear systems. In *High Per-*

formance Computing for Computational Science - VECPAR 2002, Porto, Portugal. (2003), J. M. Palma, J. Dongarra, V. Hernandez, and A. de Sousa., Eds., vol. 2565 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 436–451.

[5] SCHWAN, K., AND PLALE, B. dQUOB: Efficient queries for reducing end-to-end latency in large data streams. In *High Performance Distributed Computing (HPDC-9), August 2000* (2000).

[6] SESHAGIRI, L., SOSONKINA, M., AND ZHANG, Z. Electronic structure calculations and adaptation scheme in multi-core computing environments. In *Computational Science - ICCS 2009, 9th International Conference, Baton Rouge, LA, USA, May 25-27, 2009, Proceedings, Part I* (2009), G. Allen, J. Nabrzyski, E. Seidel, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, Eds., vol. 5544 of *Lecture Notes in Computer Science*, Springer, pp. 3–12.

[7] SESHAGIRI, L., SOSONKINA, M., AND ZHANG, Z. Exploring tuning strategies for quantum chemistry applications. In *In Proc. 2009 Int'l Workshop on Automatic Performance Tuning (iWAPT-2009), Tokyo, Japan, Oct 1-2, 2009* (2009).

[8] SESHAGIRI, L., WU, M., SOSONKINA, M., ZHANG, Z., GORDON, M. S., AND SCHMIDT, M. W. Enhancing adaptive middleware for quantum chemistry applications with a database framework. In *Int'l Parallel and Distributed Processing Symposium (IPDPS 2010), 11th Workshop on Parallel and Distributed Scientific and Engineering Computing, Atlanta, GA, Apr. 2010* (2010).

[9] SOSONKINA, M. Runtime adaptation of an iterative linear system solution to distributed environments. In *Applied Parallel Computing, PARA 2000.* (2001), T. Sørevik, F. Manne, R. Moe, and A. H. Gebremedhin., Eds., vol. 1947 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 132–140.

[10] S.STORIE, AND SOSONKINA, M. Packet probing as network load detection for scientific applications at runtime. In *Proc. 18th International Parallel & Distributed Processing Symposium (IPDPS), Santa Fe, NM.* (2004), IEEE Computer Society. 10 pages.

[11] TAPUS, C., CHUNG, I.-H., AND HOLLINGSWORTH, J. K. Active Harmony: Towards automated performance tuning. In *In Proceedings from the Conference on High Performance Networking and Computing* (2003), pp. 1–11.

[12] USTEMIROV, N., SOSONKINA, M., GORDON, M. S., AND SCHMIDT, M. W. Dynamic algorithm selection in parallel GAMESS calculations. In *Proc. 2006 International Conference on Parallel Processing Workshops, Columbus, OH* (2006), IEEE Computer Society, pp. 489–496.

[13] WIEDER, P., ZIEGLER, W., AND KELLER, V. IANOS – efficient use of hpc grid resources. *ERCIM News 2008*, 74 (2008).